A-13

定数畳み込みを対象としたC コンパイラのランダムテスト
Random Testing of C Compilers Targeting Constant Folding

武田 直哉 † 粟津 裕亘 ‡ 石浦 菜岐佐 †
Naoya Takeda Hironobu Awazu Nagisa Ishiura

1　はじめに

組み込みシステムの厳しい性能・電力比を達成するために, 特定用途向けのプロセッサが数多く開発されるのに伴い, それだけのコンパイラの開発も必要になる. コンパイラはソフトウェア開発の基盤として高い完成度を求められるソフトウェアであり, 徹底的なテストが必要になる.

コンパイラのテストは, 数千から数十万のプログラムから成るテストスイートを用いて行われるが, それを経てもなおバグが潜在することがある. そのため, 時間の許す限りランダムテスト (ランダムに生成したプログラムによるテスト) を行い, 開発者やテストスイートが想定しないようなバグを検出する努力が行われる.

文献 [1] では, 整数型の算術式を対象とした C コンパイラのランダムテストを提案している. ランダムな算術式を含むテストプログラムを生成し, これをコンパイルし実行した結果を期待値と比較することによりコンパイラのテストを行う. この方法により x86 用 GCC-4.1.2 の定数畳み込み処理に関するバグを検出しているが, 処理系 (ターゲット) に依存する部分があるため, これをそのまま他のターゲットに適応することができていなかった.

そこで本稿では, [1] のランダムテストにおいて処理系定義部を独立させ, 複数のターゲットに対応できるようする. 具体的には, 各整数型で表現可能な数値の範囲と, 右シフト演算の動作を独立したファイルで定義する. 本手法を用いて 16 ビットプロセッサ H8300 と 32 ビットプロセッサ M32R の GCC4.4.1 のテストを行った結果, バグを各 1 つ検出することができた.

2　定数畳み込みを対象としたランダムテスト

2.1　定数畳み込み

定数畳み込みはコンパイラの最適化の一種であり, 図 1 のように, 定数式の計算をコンパイル時に行うというものである. 単純な処理であるが, ターゲットに依存して処理が変わることに留意する必要がある. 例えば, 図 1 で unsigned int 型が 32 ビットの場合には y=65536 となるが, 16 ビットの場合にはラップアラウンドが生じるために y=0 となる. また, volatile 修飾 (その変数が関わる部分の最適化を制御する) があるとコンパイル結果に誤りが生じやすいことが報告されている [2].

2.2　C コンパイラのランダムテスト

本研究では, [1] と同様に図 2 に示す流れによってテストを行う. ランダムテストジェネレータが, テストプログラムを生成する. 期待値はジェネレータが計算し, テスト

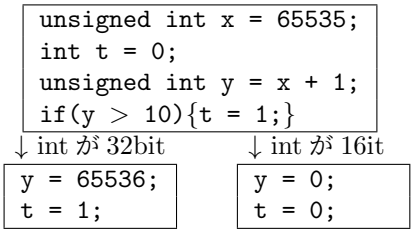

図 1: 定数畳み込み

プログラム自身の中で計算結果と比較する (一致しない場合はエラーとする). このテストプログラムを必要な最適化オプションを用いてコンパイル・実行し, エラーが発生するかどうかを調べる. この処理を時間の許す限り繰り返す. もしエラーが発生した場合には, そのプログラムを保存し, 原因の解析を行う.

プログラムと期待値の生成においては, C 言語仕様における未定義, 未規定, 処理系定義に留意する必要がある. 動作が未定義とは, オーバーフローや零除算等に対して処理系が予測不能な値を与えるということであり, 未定義動作を含むプログラムはテストに含めるべきでない. 未規定とは関数の引数や式中の部分式の評価順等が言語仕様上一意に定められていないことであり, 期待値がこれに依存して変化するプログラムはテストに含めるべきでない. 処理系定義は, int 型で表現できる数値の範囲のように, 処理系毎に定義されるものであり, テストプログラムの期待値はこの定義に従って計算しなければならない.

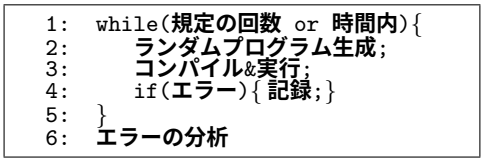

図 2: ランダムテストの流れ

3　複数ターゲットへの対応

C 言語の処理系定義のうち, 本研究で生成するプログラムが影響を受けるのは,

(1) 各整数型で表現可能な数値の範囲
(2) 左オペランドが負の場合の右シフトの結果
(3) 代入式において, 右辺の値が左辺の型で表現不可能な数値の場合に代入される値

である. (1) は, 符号無し変数の演算でのラップアラウンド, 符号付き変数の演算でのオーバーフロー, 整数拡張や算術型変換での結果の型等に影響する.

本稿では, (1)(2) に関しては, 定数や関数の形で処理系依存記述として独立させ, テストジェネレータ本体をター

† 関西学院大学 理工学部 情報科学科
‡ 関西学院大学 大学院理工学研究科 情報科学専攻

ゲット独立とする．(3) に関しては, 未定義・未規定と同様にテストの対象としないものとする．

処理系定義の記述例 (M32R 用) を図 3 に示す．1∼15 行目は, 各整数型の最小値と最大値を, 18∼24 行目は, 右シフト演算の動作を定義している．ターゲットを変更するときは, この部分のみを書き換えればよい．

```
1:  our $SCHAR_MAX = 127;
2:  our $SCHAR_MIN = -128;
3:  our $UCHAR_MAX = 255;
4:
5:  our $SHRT_MAX = 32767;
6:  our $SHRT_MIN = -32768;
7:  our $USHRT_MAX = 65535;
8:
9:  our $INT_MAX = 2147483647;
10: our $INT_MIN = -2147483648;
11: our $UINT_MAX = 4294967295;
12:
13: our $LONG_MAX = 2147483647;
14: our $LONG_MIN = -2147483648;
15: our $ULONG_MAX = 4294967295;
16:
17: #右シフトで左オペランドが負の場合
18: sub SHR
19: {
20:     my $l_value = shift;
21:     my $r_value = shift;
22:     my $tmp = $l_value / (2 ** $r_value);
23:     #マイナス方向に切り捨てる
24:     return &floor($tmp);
25: }
```

図 3: 処理系定義の記述例

4 実験結果

提案手法に基づくランダムテストのシステムを Perl5 で実装した．動作環境は, Windows 上の cygwin, Mac OSX, Linux である．

4.1 H8300 用 GCC のテスト

16 ビットプロセッサ H8300 用のコンパイラ GCC-4.4.1 (Linux) のテストを行った．最適化オプション-O1 でテスト開始したところ, 1 時間で 19,736 個のプログラムを生成・実行し, このうち 46 個でコンパイラの異常終了 (計算結果の不一致ではなくコンパイラのクラッシュ) が発生した．図 4 は, エラーを引き起こしたプログラムの 1 つを最小化したものである (他の 45 個のプログラムも同様の結果となった)．7 行目の式の期待値は 0 だが, オプション-O1 以上でコンパイルするとコンパイラが異常終了し, コンパイルできなかった．なお最適化オプションなしでは正常にコンパイルが行え期待値を得ることができる．

```
1:  #include <stdio.h>
2:
3:  int main (void)
4:  {
5:    volatile int x = 1;
6:
7:    long test = 7 * ((long)1 >> 1 / x);
8:
9:    if(test == 0){
10:       printf("@OK@¥n");
11:    }else{
12:       printf("@NG@¥n");
13:       printf("%d¥n",test);
14:    }
15:    return 0;
16: }
```

図 4: H8300 用 GCC のバグを検出したプログラム

4.2 M32R 用 GCC のテスト

32 ビットプロセッサ M32R 用のコンパイラ GCC-4.4.1 (Linux) のテストを行った．最適化オプション-O1 でテストを開始したところ, 1 時間で 21,124 個のプログラムを生成・実行し, 9 個のエラーを検出した．図 5 は, そのうち 1 つを最小化したものである (他の 8 個のプログラムも同様の結果となった)．7 行目の式の期待値は −40000 だが, オプション-O1 以上でコンパイル, 実行すると 25536 となった．なお, 最適化オプションなしでは期待値を得ることができる．また変数から volatile を削除すると最適化オプションありでも正常に動作する．

最適化オプションを指定したときに正常に動作しないのは, 整数拡張に問題があるためと推測される．7 行目の乗算において short 型の VAR1 の値は int 型に整数拡張されるべきだが, 最適化オプションを指定すると short 型のままで計算され, ラップアラウンドが生じていると考えられる．

```
1:  #include <stdio.h>
2:
3:  int main (void)
4:  {
5:    volatile short VAR1 = -1000;
6:
7:    long test = (40*VAR1);
8:
9:    if(test == -40000){
10:       printf("@OK@¥n");
11:    }else{
12:       printf("@NG@¥n");
13:       printf("%d¥n",test);
14:    }
15:    return 0;
16: }
```

図 5: M32R 用 GCC のバグを検出したプログラム

5 結論

本稿では, C コンパイラの定数畳み込み処理を対象とするランダムテストにおいて処理系定義部を独立させ, 複数のターゲットにも対応できるようにした．本手法により, H8300 用コンパイラ GCC-4.4.1 および M32R 用コンパイラ GCC-4.4.1 のバグを 1 つずつ検出することができた．

今後の課題として, バグを検出したプログラムの最小化の自動化, 浮動小数点を含むランダムテストの作成などが挙げられる．

謝辞

本研究関して有益な御援助を頂いた株式会社 SRA の引地信之様, SRA OSS, Inc. の矢吹洋一様に感謝致します．

参考文献

[1] 粟津裕亘, 石浦菜岐佐: “算術式の最適化を対象とした C コンパイラのランダムテスト,” 信学技報, VLD2008-127, pp. 7–10 (Mar. 2009).

[2] Eric Eide and John Regehr: “Volatiles Are Miscompiled, and What to Do about It,” in *Proc. ACM Int. Conf. on Embedded Software*, pp. 255–264 (Oct. 2008).